NPS72An75111

Library

NAVAL POSTGRADUATE SCHOOL

Monterey, California

DESIGN CONSIDERATIONS FOR THE
NPS SIGNAL PROCESSING AND DISPLAY LABORATORY
MULTIPROCESSING OPERATING SYSTEM

by

B. E. Allen
and
G. L. Barksdale, Jr.

November 1975

Approved for public release; distribution unlimited.

Prepared for: Naval Electronics Systems Command (Code 320)
Washington, D. C.
ASW Special Projects Office
Washington, D. C.

FEDDOCS
D 208.14/2:NPS-72AN75111

NAVAL POSTGRADUATE SCHOOL
Monterey, California

Rear Admiral Isham Linder
Superintendent

Jack R. Borsting
Provost

The work reported herein was supported by the Naval Electronics Systems Command (Code 320) and the ASW Special Projects Office.

Reproduction of all or part of this report is authorized.

This report was prepared by:

UNCLASSIFIED

SECURITY CLASSIFICATION OF THIS PAGE *(When Data Entered)*

REPORT DOCUMENTATION PAGE		READ INSTRUCTIONS BEFORE COMPLETING FORM
1. REPORT NUMBER NPSAn75111	2. GOVT ACCESSION NO.	3. RECIPIENT'S CATALOG NUMBER
4. TITLE *(and Subtitle)* Design Considerations for the NPS Signal Processing and Display Laboratory Multiprocessing Operating System		5. TYPE OF REPORT & PERIOD COVERED Technical Report 6. PERFORMING ORG. REPORT NUMBER
7. AUTHOR*(s)* B. E. Allen and G. L. Barksdale, Jr.		8. CONTRACT OR GRANT NUMBER*(s)*
9. PERFORMING ORGANIZATION NAME AND ADDRESS Naval Postgraduate School Monterey, California 93940		10. PROGRAM ELEMENT, PROJECT, TASK AREA & WORK UNIT NUMBERS
11. CONTROLLING OFFICE NAME AND ADDRESS		12. REPORT DATE November 1975 13. NUMBER OF PAGES 21
14. MONITORING AGENCY NAME & ADDRESS*(if different from Controlling Office)*		15. SECURITY CLASS. *(of this report)* UNCLASSIFIED 15a. DECLASSIFICATION/DOWNGRADING SCHEDULE

16. DISTRIBUTION STATEMENT *(of this Report)*

Approved for public release; distribution unlimited.

17. DISTRIBUTION STATEMENT *(of the abstract entered in Block 20, if different from Report)*

18. SUPPLEMENTARY NOTES

19. KEY WORDS *(Continue on reverse side if necessary and identify by block number)*

real-time, operating systems, PDP 11/50, timesharing, UNIX

20. ABSTRACT *(Continue on reverse side if necessary and identify by block number)*

The design and implementation of MUNIX, a tightly-coupled symmetric multiprocessing PDP 11 based operating system providing real-time, interactive, and background processing facilities in a hierarchical memory environment is described. MUNIX is a variant of UNIX, an operating system for the PDP 11 developed at Bell Laboratories.

The three major design goals of the system were:

DD FORM 1 JAN 73 1473 EDITION OF 1 NOV 65 IS OBSOLETE
S/N 0102-014-6601 |

UNCLASSIFIED

SECURITY CLASSIFICATION OF THIS PAGE *(When Data Entered)*

20. (cont)

(1) support for processes capable of real-time interaction with several dynamic graphics display units, an array processor, and a multi-channel A/D converter;
(2) interactive and background processing facilities to support program development; and,
(3) management of the hierarchical storage created by the mix of shared and private memories of various speeds.

The resulting MUNIX system provides an effective mechanism for resource sharing in a laboratory environment and is the basis for protected real-time operation in a multi-user system.

Design Considerations for the NPS Signal Processing and Display Laboratory Multiprocessing Operating System

B. E. Allen and G. L. Barksdale, Jr.
Computer Science Group
Naval Postgraduate School
Monterey, California 93940

ABSTRACT

The design and implementation of MUNIX, a tightly-coupled symmetric multiprocessing PDP 11 based operating system providing real-time, interactive, and background processing facilities in a hierarchical memory environment is described. MUNIX is a variant of UNIX, an operating system for the PDP 11 developed at Bell Laboratories.

The three major design goals of the system were:
(1) support for processes capable of real-time interaction with several dynamic graphics display units, an array processor, and a multi-channel A/D converter;
(2) interactive and background processing facilities to support program development; and,
(3) management of the hierarchical storage created by the mix of shared and private memories of various speeds.

The resulting MUNIX system provides an effective mechanism for resource sharing in a laboratory environment and is the basis for protected real-time operation in a multi-user system.

Keywords: Real-time, operating systems, PDP 11/50, time-sharing, UNIX

CR Categories: 3.80, 4.32, 6.22

INTRODUCTION

The flavor of a computer operating environment is often derived from its most stringent processing requirements. Thus, real-time systems tend to provide very sparse program development facilities, terminal systems often overlook adequate background processing mechanisms or real-time support, and batch systems tend to avoid all interactive tasks. However, we assert that any system seeking to support the active development of real-time tasks needs the best available software engineering tools; the MUNIX operating system is the result of our efforts to provide this environment.

Equipment Configuration

The configuration of the Signal Processing and Display Laboratory is shown in Figure 1. The real-time system can be viewed as a three bus ensemble, with the respective functions of data acquisition, signal processing, and display. When bus cycles are not required by real-time processes, the data acquisition and display busses support program development activities. The display system includes a 256K word fixed head disk, a Ramtek color display, a Tektronix 4014 display with enhanced graphics, a Vector General 3D3I system, a Hughes Conographic console, a data tablet, a Versatek printer/plotter, and an EPC graphic recorder. Peripherals for the data acquisition controller include both large (96M words) and small (2.5M words) disk systems, magnetic tapes, a card reader, a line printer, and a sixteen line

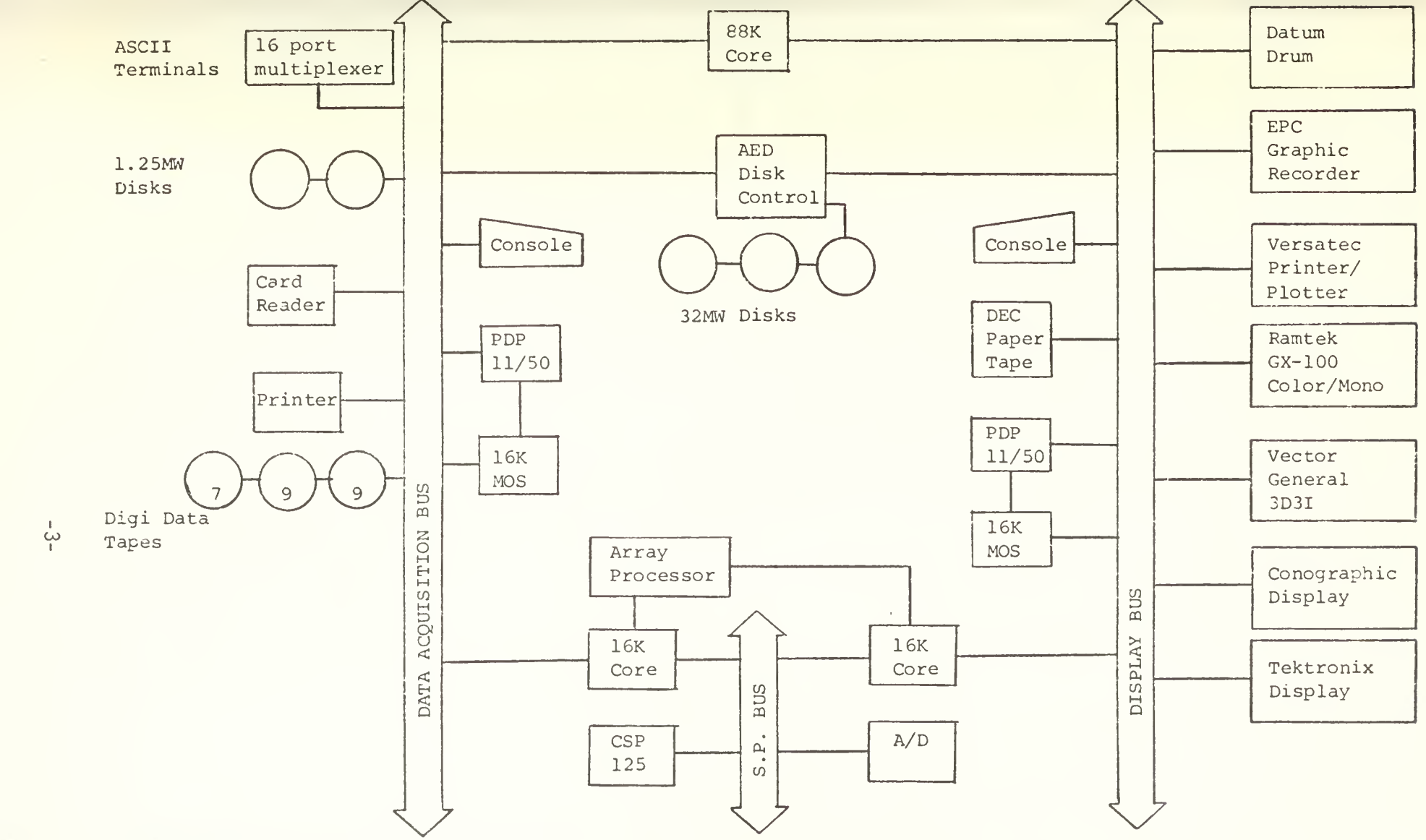

Figure 1. NPS Signal Processing and Display Laboratory Equipment Configuration

programmable terminal multiplexer. Dual ported core memory (88K words) is accessible from either UNIBUS. The signal processing subsystem consists of a CSP 125 controller with 4K words of 125 nanosecond memory, an array processor, and two 16K word banks of three ported memory.

Operational Factors

The MUNIX operating environment is a university laboratory engaged in research and educational use of computer graphics, signal processing, operating systems, and hybrid computing. Although several operating systems [5,6,7,10] are available for use with the PDP-11 computer, each lacks capablity in some dimension which appears important to the present range of applications. We were thus faced with the alternative of maintaining several operating systems for use with the various applications (with the attendant equipment scheduling and program conversion problems) or developing a unified operating environment with subsystems which provide the required specialized support when it is needed. In the paragraphs which follow, we present the key features of the multi-function operating system (MUNIX) which provides a support environment for the development, maintenance, and operation of real-time programs.

Since program development is a major portion of our workload, we sought a multiprogramming operating system which would provide us with interactive terminal support, a hierarchical file system, and a full complement of program development software (editor, compilers, assembler, and

utilities). Three other major considerations were the ease of interfacing new devices to the operating system, the system's support of extended addressing (memory management), and the availability of source code. Using these criteria, the UNIX [10] operating system was chosen as the basis for program development support.

Once the decision was made to utilize UNIX as the basic operating environment, attention could be focused on the technical problems associated with providing to our community of users an environment which facilitates the implementation and debugging of real-time processes. The problem of providing user access to the full set of peripherals on both processors while seeking to dynamically balance the UNIBUS and processor loads and provide real-time support for the display and signal processing tasks led to the development of the symmetric multi-processing operating system discussed in subsequent sections.

Architectural Considerations

A single bus architecture such as the PDP 11 is not a favorable environment for multiprocessing because each processor can only communicate with the peripherals and memory on its own bus. One (expensive) method of solving this problem is to buy peripherals which are multi-ported and thus capable of communicating with more than one bus; another approach uses special purpose bus switches which toggle one peripheral between the two busses. In light of the presently available switch technology, this solution was

also rejected as economically infeasible.

Except for the disk storage unit and core memory, MUNIX swaps processes between the two processors in order to meet peripheral access requirements. Thus, the access problem was solved without the benefit of special purpose bus switches or multi-ported peripherals. Bus traffic is spread across both busses and only those processes which must access devices on both busses will incur the shared access overhead. We believe this solution to be appropriate for use in real-time systems in which the real-time device access can be confined to a specific bus.

Another problem which required attention was the controlled utilization of the several kinds of memory available in the system. In a system with a small (18-bit) address range, each page frame is extremely valuable; thus, it is desirable for special purpose memory to be available for general allocation whenever it is not required for real-time operation.

MULTIPROCESSING

MUNIX is a tightly-coupled symmetric multiprocessor operating system which is designed to provide a mechanism for bus and processor load balancing as well as a uniform user interface to a wide varity of system peripherals. In general, the design is similar to other multiprocessor systems [1,2,4,9] -- the single copy of the system residing in shared memory uses P and V operators [8] for synchronization. The processors are completely independent, each doing its own user process selection from a single ready process list. In order to facilitate processor identification, the hardware was modified so that the three unused bits in the processor status word (bits 8-10) contain a unique processor identifier.

As Figure 1 indicates, the hardware is not symmetrical with respect to I/O devices on the two UNIBUS's. The most important devices, bulk memory and the large disk storage unit, are dual-ported and can be accessed by processes running on either processor; all other devices are single-ported and may only be accessed by their host processor. Most multiprocessor operating systems avoid this problem by employing I/O controllers or channels which communicate with all processors. Since the concept of a UNIBUS communicating with more than one processor is foreign to the PDP-11 hardware design, another solution to the I/O control problem had to be found for MUNIX.

Processor Affinity

Because the distribution of devices across the two busses in the system is not symmetric, the concept of processor affinity was introduced into MUNIX. Processor affinity may be requested by the user or the system, and may be permanent or temporary, and may have be advisory or mandatory status. The use of each of these types of processor affinity is discussed in the following paragraphs.

It is not feasible to make a priori determination of the I/O device requirements of all processes nor is it desirable to limit a process to the peripherals attached to only one bus; therefore, MUNIX supports dynamic process-processor affinity. It appears desirable to determine device availability in a truly dynamic fashion, with each processor initiating all user I/O requests as though all devices were available on both buses. Only when this access attempt had failed (as indicated by an addressing error) would the process be passed to the other processor where the I/O would again be initiated. Unfortunately, much of the system I/O set-up has to be repeated by the second processor, and in addition, the time for the hardware to discover and report device nonexistence varies between five and ten microseconds and stops all bus activity. This scheme would have made device reconfiguration simple but was abandoned because of excessive system overhead.

In the current implementation, MUNIX maintains a dynamic configuration table which lists the devices on each UNIBUS

(the multi-ported devices are in both lists). When a process requests an I/O operation, MUNIX determines if the required device can be accessed by the processor which is currently servicing the process. If not, a temporary processor affinity flag is set and the user task is suspended. When next scheduling this task for a processor, the system uses the temporary affinity flag to insure that the correct processor is chosen. When the I/O operation is completed, the temporary affinity flag is cleared and the process is once again a candidate for execution by either processor.

In order to decrease scheduling overhead, a permanent processor affinity flag is provided for processes which do large amounts of I/O to devices which are accessible from only one bus. This flag is set by the user process via a system call which specifies the device required by the user; MUNIX uses the configuration table to translate this request into the appropriate permanent affinity flag. Once set, this advisory flag is used by the processor scheduler. As long as more than one process is ready for execution, a process with the permanent affinity flag will only be scheduled on the desired processor. If only one process is ready, however, it will be executed by either processor since the alternative is an idle processor. A process with the permanent affinity flag set will be temporarily switched in order to access an I/O device on the other UNIBUS. Neutral processor affinity is the default for non-real-time processes.

MUNIX solves the problem of accounting for a separate stack for each processor in a rather clean, straight-forward manner. With each user process the system associates 1024 bytes of storage. This region contains system per-process data and the system stack for this process. When attention is switched from one process to another, it is sufficient to simply change the system stack segment register to point to the header area of the new process and reload the stack pointer; the state of that process is thereby completely restored. This scheme is sufficiently general to allow capture of a nested interrupt state as well as the normal user state. Since this elegant scheme was used in the UNIX monoprocessing system [10], no change was required to allow multiprocessing.

MEMORY HIERARCHY MANAGEMENT

As indicated in Figure 2, the system primary memory is rather unusual. Each processor's memory space consists of 16K words of 450 nanosecond MOS memory which is private, 16K words of 850 nanosecond core memory which is shared with the CSP array processor, and 88K words of 800 nanosecond core which is shared between the PDP 11/50 processors. The operating system occupies 24K words of the shared memory.

MUNIX provides each user program with a virtual memory of up to 32K words which is divided by the memory management hardware into eight pages of 4K words each. Having thirty-two page frames at its disposal, the memory management software utilizes a working set algorithm [3] for page replacement.

When there are no real-time processes active, the memory manager uses both private and shared page frames uniformly except for the restriction that pages from one process can not reside simultaneously in both private memories. Obviously, a process which has one or more pages in a private memory can only be executed by one processor. If this process must use the other processor to access a peripheral device, any pages in private memory are moved to the shared memory before the temporary affinity flag is set. Since this move involves significant overhead (the PDP 11 has no block move instruction), the system attempts to avoid this situation by assigning the private memory to processes which have the permanent affinity flag set wherever possible.

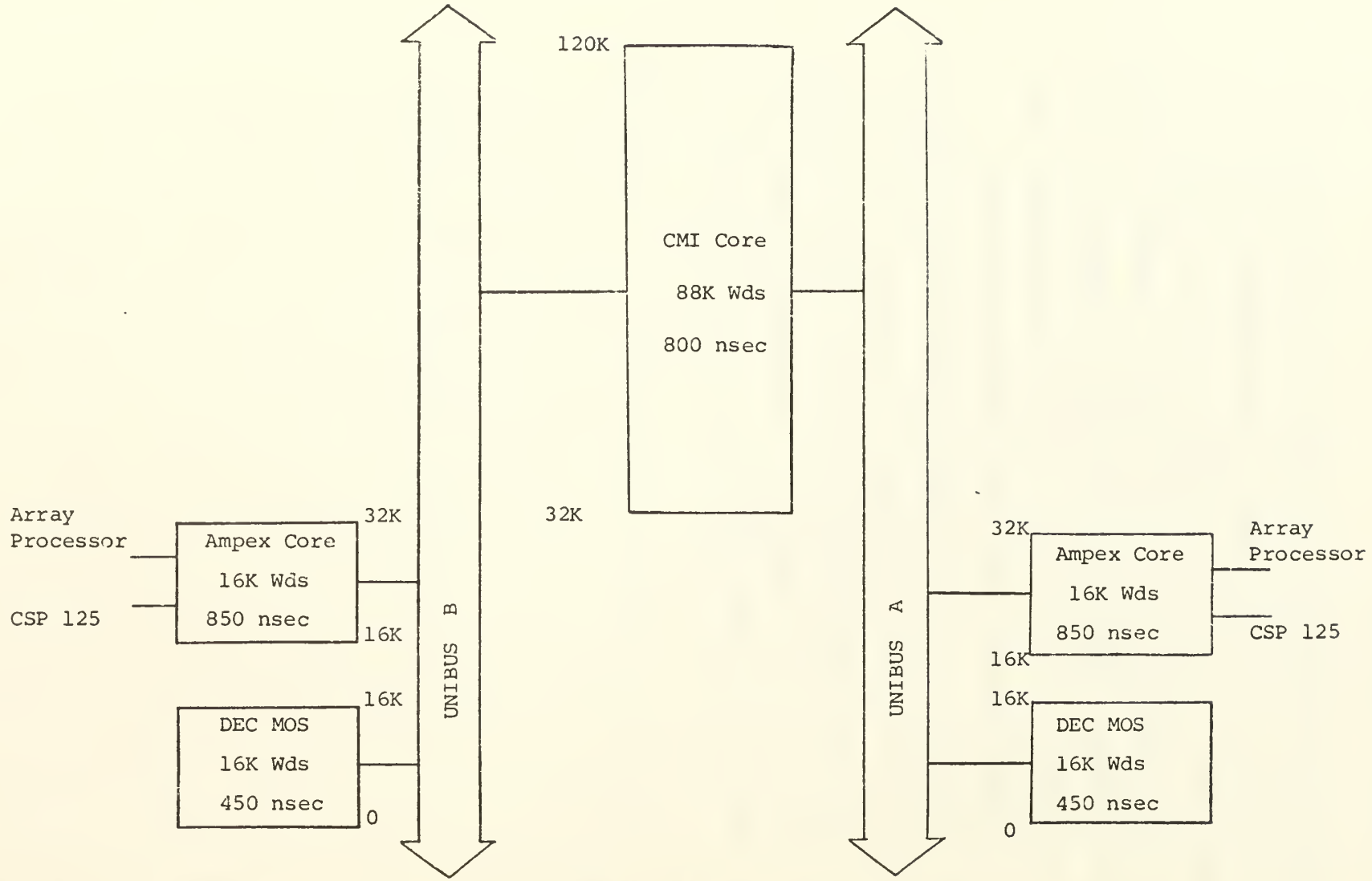

Figure 2. Memory Hierarchy

When a process is granted real-time status, the memory management software removes the pages of all other processes from the private memory of the desired processor. If page frames are available and the process status warrants, these pages are moved to shared memory; otherwise, they are moved to the swap file. Once the private memory is cleared, all of the pages of the new real-time process are moved from wherever they reside (the other private memory, shared memory, or the swap file) and locked so that they will never be swapped. This move may involve a transfer to and from the swap file if the page happened to reside in the wrong private memory. Once moved, the real-time process is executed from the fastest memory on the system.

PROCESS CONTROL

MUNIX supports three types of processes: foreground (timeshared), background (batch), and real-time. Since the control and capability of the first two types of processes have been reported elsewhere [10], this paper will concentrate on real-time process support.

One of the primary functions of the Signal Processing and Display Laboratory is to support signal acquisition and analysis. To accomplish this task, an analogue signal is digitized and then sent to the data acquisition process. This process logs the data and does some front end analysis before passing it to the CSP 125 with its array processor where the signal is processed using Fourier transform techniques. The transformed data is then passed to a real-time display process which presents the data on one or more of the graphic display devices in the system. Both the data acquisition process and the display process operate under severe time constraints. In order to support this type of computing, MUNIX provides a real-time process classification.

Real-Time Processes

All processes begin as either foreground or background. After a process starts execution, it can request, via a system call, that it become a real-time process. Since a real-time process is by definition attempting to respond to some external stimulus (device), these processes must be

executed on the processor whose UNIBUS is connected to the desired device. Therefore, every real-time process has a permanent processor affinity which is mandatory rather than advisory.

When a process requests real-time status, the system makes two limit checks. First, it determines the number of real-time processes in existence with an affinity for the desired processor. If this number is equal to a maximum value (currently one), the request is denied. Second, it checks the total amount of memory dedicated to real-time processes. If this amount plus the amount required for the requesting process is greater than a maximum value (currently 64K words), the request is denied. These limit checks enforce the system policy of not allocating system resources to real-time processes to the point of severly degrading system response to other users. Both of these restrictions are merely administrative, system is concerned although the policy of allocating only private memory to real-time processes obviously must be discarded if such processes are allowed to occupy more than 64K. The problem of multiple real-time processes competing for the same processor is solved by a dynamic system limit on the number of consecutive quanta which will be allotted to a real-time process when other processes with sufficiently high priority are waiting.

If the real-time request can be granted, the requesting process is moved into the private memory of the specfied processor (Figure 2), possibly dislocating some nonreal-time

processes. After this move, the process is locked into memory so that it is never a candidate for swapping and is given the highest possible priority. Thus, whenever a real-time process becomes ready for execution, it is preemptively allocated a processor and keeps possession of this processor until it completes or until an I/O request causes it to be blocked.

Array Processing and Real-Time I/O

As indicated in Figure 2, the upper 16K words of each processor's private memory is shared with the CSP-125. In order to reduce the overhead involved in communication between a real-time process and the CSP, this memory is made a portion of the real-time process address space. Thus, a process which wishes to communicate with the array processor need only store the data in the upper 16K words of its own address space and request the operating system to send an interrupt. Similarly, data coming from the array processor is placed directly into the address space of a real-time process and thereby saves the system overhead involved in a block move; an input data ready interrupt is also provided.

Interactive Graphics

In addition to accomplishing very efficient communication with the array processor, the method chosen for supporting real-time processes solves a problem which would have been difficult to solve in any other manner. One of the graphic display units in the system, a Vector General

3D31, is a refresh device which retrieves and interprets a display list forty times a second. The display list contains not only physical memory addresses to be used in direct memory access transfers, but also instructions which can cause the display to store information anywhere within the 32K word segment of memory which contains the display list. Although MUNIX can control segment access, there is no effective mechanism for controlling intra-segment memory references in a user's display list since the hardware applies neither memory protection nor relocation to these memory accesses.

Since the display list is quite large and complex, it is not feasible to have the operating system build a valid list from parameters supplied by the user, or verify a user's list before it is sent to the display controller. However, if the user were allowed to send an arbitrary (unchecked) display list to the display controller, MUNIX could not insure the integrity of any other process residing in the same 32K memory segment. Our solution to this problem is to require a real-time classification for all processes which use this display unit. As noted earlier, real-time processes are placed in the 32K word private memory segment and all other processes are removed from this area. Thereafter, the user process is allowed to specify the display list which the operating system sends to the display controller unchecked. The worst consequence of an invalid display list in this environment is that it may destroy the process which built the list.

Another interesting by-product of the real-time process control is a very simple method for dynamic system partitioning. Since the private memory of each processor is the low order 32K words of address space, it is very easy to separate one processor, its private memory, and its I/O devices. Once separated, this portion of the hardware can be used to run other operating systems or to test stand-alone programs. In fact, the stand-alone program or operating system can be built in the MUNIX environment, loaded into the private memory as a real-time process and then be given complete control of the hardware environment as MUNIX reverts to a monoprocessing system to continue serving its other users.

CONCLUSIONS AND FUTURES

Our major efforts to date have been toward implementation of a logically correct MUNIX system which makes only those changes to UNIX required by the structural differences of the present operating environment. The limited experience we have had to date on MUNIX has served to confirm our basic design decisions. Supporting all facets of real-time programming within a uniform environment has greatly simplified the overall system design.

With each task taking its respective place in a service hierarchy, the available system capacity can be dynamically allocated to the priority process, thereby avoiding worst case a priori resource allocation. This approach has had the further advantage of providing a vehicle for rapid (less than six months elapsed time) development of a sophisticated system by a small programming group (2 faculty, 6 graduate students).

Other completed work includes the development of a dynamic symbolic debugging tool, development of numerous on-line diagnostic packages and I/O device drivers, development of a line editor which facilitates correction of typing mistakes on the interactive level, and enhancements to the text editor, the text processor, and the linking loader. Work presently underway includes a performance measurement subsystem, several adaptive schedulers, a virtual machine monitor, and a hardened file system.

REFERENCES

[1] Alexander, M. T. The Organization and Features of the Michigan Terminal System. Proc. AFIPS 1972 SJCC, Vol. 40, AFIPS Press, Montvale, N.J., pp. 585-591.

[2] Anderson, J. P., Hoffman, S. A., Shifman, J., and Williams, R. J. D825 -- A Multiple-Computer System for Command and Control. Proc. AFIPS 1962 FJCC, Vol. 22, AFIPS Press, Montvale, N.J., pp. 86-96.

[3] Denning, P. J. The Working Set Model for Program Behavior. Comm. ACM 11,5(May, 1968),pp. 323-333.

[4] Denning, P. J. Third Generation Computer Systems. ACM Computing Surveys 3,4(Dec, 1971),pp. 175-216.

[5] Digital Equipment Corporation. The DOS/BATCH Handbook, 1974.

[6] Digital Equipment Corporation. RSTS-11 System User's Guide, 1974.

[7] Digital Equipment Corporation. RSX-11D Concepts and Capabilities, 1974.

[8] Dijkstra, E. W. The Structure of T.H.E. Multiprogramming System. Comm. ACM 11,5(May, 1968),pp. 341-346.

[9] Organick, E. I. The Multics System: An Examination of Its Structure. MIT Press, Cambridge, Mass. 1972.

[10] Ritchie, D. M. and Thompson, K. The UNIX Time-Sharing System. Comm. ACM 17,7(July, 1974)pp. 365-,375.

INITIAL DISTRIBUTION LIST

	Copies
Dean of Research Code 023 Naval Postgraduate School Monterey, CA 93940	1
Defense Documentation Center Cameron Station Alexandria, Virginia 22314	2
Library (Code 0212) Naval Postgraduate School Monterey, CA 93940	2
Library (Code 72) Naval Postgraduate School Monterey, CA 93940	1
ASW Special Projects Office Attn: Mr. Robert Bryant Department of the Navy Washington, D. C. 20360	1
W. R. Church Computer Center Naval Postgraduate School Monterey, CA 93940	1
Naval Electronics Systems Command (ELEX 320) Department of the Navy Washington, D. C. 20360	1
Naval Electronics Laboratory Center Library 271 Catalina Boulevard San Diego, CA 92152	1
Mr. Michael Griswold FCDSSA 270 Catalina Boulevard San Diego, CA 92147	1
Mr. Marvin Denicoff Office of Naval Research Department of the Navy Arlington, Virginia 22217	1
Prof. G. L. Barksdale, Jr. (Code 72Ba) Computer Science Group Naval Postgraduate School Monterey, CA 93940	50

U171469

DUDLEY KNOX LIBRARY - RESEARCH REPORTS
5 6853 01071183 1